TIME SERIES MONITORING SYSTEM

Inventors: Joseph Billock

2387 N. Garfield Ave.

Altadena, CA 91001

USA

US Citizen

Ian Swett

1465 N. Mar Vista Ave.

Pasadena, CA 91104

USA

US Citizen

Eric Bax

2241 Santa Rosa Ave.

Altadena, CA 91001

USA

US Citizen

CROSS REFERENCE TO RELATED APPLICATIONS

This invention relates to Provisional Application 60/451,490.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

Not Applicable.

REFERENCE TO A SEQUENCE LISTING, A TABLE, OR A COMPUTER PROGRAM LISTING COMPACT DISK APPENDIX

Not Applicable.

BACKGROUND OF THE INVENTION

A time series is a sequence of data indicating values over time. One example is the sequence of daily high temperatures in a city. Another example is the sequence of prices paid for a commodity over time.

One tool for examining time series is a window, which is a subsequence. A time window includes the values associated with a time period. A value window includes a specified number of values. The subsequence specified by a moving window changes over time for a moving time window and changes as values are received for a moving value window. For example, a five-minute moving window includes the values for the past five minutes.

Statistics over windows are useful to monitor time series. For example, the five-minute moving average of a time series is the average of values in the five-minute moving window. Since the values in moving windows change over time, window statistics also change over time. The straightforward method to re-compute a window statistic is to access all values in the window and compute the statistic directly. Online computation is another method. In online computation, a statistic value is computed by modifying the previous value of the statistic to account for values that expired from the

window and values added to the window since the previous computation. For example, consider re-computing the sum of a hundred-value moving tick window when a new value is received. The straightforward method is to take the sum of the new value and the ninety-nine most recent previous values. The online method is to take the previous sum, subtract the oldest value used in the previous sum, and add the new value. The straightforward method requires about a hundred mathematical operations; the online method requires two.

When a moving window is inserted on a time series, there can be a time delay before the window becomes valid. For example, consider adding a five-value moving window to a time series. If the window statistics computation only uses values received after the window is formed, then the statistics are not valid until five new values are received. On the other hand, if historical values are available, then they can be used to compute the statistics. As a result, the window becomes valid earlier. For example, if there are values for a two-value moving window and a five-value moving window is inserted, then it is possible to have valid statistics after three new values.

One tool to monitor a set of time series is a persistent query. The persistent query contains an event condition and a payload specification. A system that executes a persistent query sends the specified payload as output if the event condition holds. The event condition may involve statistics over windows. For example, a persistent query could include the event condition:

five-day moving average temperature in Anaheim is more than 20 degrees higher than the ten-day moving average temperature in St. Louis

and the payload specification:

latest price for a flight from St. Louis to Anaheim.

There are many uses for a system to monitor time series data. In financial market trading, it is useful to monitor prices of multiple commodities or of the same commodity on different exchanges in order to trade when conditions indicate likely profit. In financial market-making, it is useful to monitor prices and volume in order to adjust bid-ask spreads in

response to changes in volatility. For an electrical power provider, it is useful to monitor power usage and availability over time at different locales in order to produce and route power efficiently. Some desirable features of a time series monitoring system include the following.

() Support high data throughput with low response time.

() Support multiple input time series.

() Execute persistent queries.

() Support dynamic management of persistent queries, i.e., support insert and delete of persistent queries without halting input and persistent query execution.

() Support dynamic management of windows.

() Perform online computation of statistics.

() Use historical values in present windows to help populate inserted windows.

Previous technologies have some of these features, but none has all. One previous technology with some of these features is a database. Another is online statistics software. Yet another is a system that combines online statistics software with a database.

A database can be configured to support multiple time series, execute persistent queries, and support dynamic management of persistent queries and windows, as follows. For each time series, use a database table to store each value in a record that also has a timestamp field which indicates when the value is received. For each persistent query, form a database trigger that executes a database query. The query encodes the condition and payload specification. Use database-supplied functions to compute statistics in the condition. For example, to compute the five-minute moving

average of a time series, apply the database-supplied average function to the values with timestamps indicating receipt within the past five minutes. A shortcoming of using the database in this way is that the database-supplied functions do not perform online computation of statistics. Hence, response time suffers.

There is software that performs online computation of statistics. Some of this software supports high data throughput with low response time and executes persistent queries. However, this software is special-purpose; it does not support dynamic management of persistent queries, support dynamic management of windows, and use historical values in present windows to help populate inserted windows.

It is possible to build a system by combining online statistics software with a database. The statistics software recieves time series values, computes statistics, and sends the statistics to the database. The database executes persistent queries. The system does not support dynamic management of persistent queries, support dynamic management of windows, and use historical values in present windows to help populate inserted windows. Even though the database alone supports these features, the system as a whole does not. The statistics software lacks these features, and both the database and the statistics software would need these features for the system as a whole to have them.

SUMMARY OF THE INVENTION

The present invention, a system to monitor time series, successfully combines some of the flexibility of a database with the speed of online statistics computation. The system executes persistent queries on multiple input time series, successfully handling high data throughput with low response time. The system supports dynamic management of time series, of windows in time series, and of persistent queries. Also, the system can use historical values in present windows to help populate inserted windows.

BRIEF DESCRIPTION OF THE DRAWING

Figure 1 is a diagram of the time series monitoring system.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

The embodiment described here is a time series monitoring system implemented as software in the Java computer language. Refer to Figure 1 for a diagram showing system parts and some of their relationships. This description begins with an introduction to system parts and their roles. Next, there is a description of how the system responds to an input value from a time series. Next, there is a description of persistent query syntax and parsing. Then processes used by the system to accomplish various functions are described in a step-by-step fashion.

Parts

The time series monitoring system can be implemented using a software class called an engine. The engine comprises a filter, a sequencer, a set of registers, and a set of persistent queries. The engine also coordinates methods to insert data into the system, to execute persistent queries, and to dynamically manage registers, windows, and persistent queries.

Each time series monitored by the system corresponds to a register. Each register has a ring buffer to store time series values. Each register may contain one or more windows. Each window is associated with a subset of the time series. Each window maintains statistics for the associated subset of the time series.

Each persistent query comprises an event condition and a payload specification. The event condition is a boolean function that may depend on statistics from multiple registers. The payload specification expresses which data are to be output by the engine when the event condition is evaluated as true. For each persistent query, there is a set of associated registers called trigger registers. The engine evaluates the event condition in response to receiving a value from any time series corresponding to a trigger register.

Response to an Input from a Time Series

The engine receives as input a stream of labeled data, in which each value is associated with a label indicating the time series to which the value belongs. For each input value, the filter uses the label to determine whether the value belongs to a time series monitored by the system. If not, then the value is discarded. If so, then the engine forms a point that

comprises the value, a timestamp, and a unique id number supplied by a sequencer. Then the engine sends the point to the register that corresponds to the time series to which the value belongs.

The register installs the point in the buffer of the register. Next, each window in the register updates the statistics for the window using online computation, accounting for the new point and for any points that have expired from the window since the previous statistics computation. Then the register adjusts the buffer, discarding points that have expired from all windows, growing the buffer if needed to accommodate new points, and shrinking the buffer if needed to accommodate storage needs of other buffers.

Then the engine executes each persistent query for which the register is a trigger. Each persistent query comprises an event condition and a payload specification. The engine evaluates the event condition. If the event condition is true, then the engine creates a payload according to the payload specification and delivers the payload as output.

Persistent Query Syntax and Parsing

A persistent query may be expressed using a query string. The query string syntax is similar to SQL, the standard database query language. The format for a query string is as follows.

SELECT <columns> FROM <trigger registers> WHERE <condition>

Columns

The columns encode the payload specification. Column syntax is as follows.

<columns> = REGISTER.WINDOW.NAME [[, REGISTER.WINDOW.NAME [, ...]]

This is a comma-separated list of identifiers of the form REGISTER.WINDOW.NAME where NAME is the name of a statistic (like COUNT, SUM, MEAN, STDEV), one of (VALID, ALL, LAST, VAL), or a number. VALID is true when the window is valid (has enough points in it or enough time in it) and false if not. ALL indicates the set of points that entered the window since the last time the event condition evaluated as true. If the window has received a new

point since the event condition last evaluated to true, then LAST indicates the newest point in the window, and VAL is the time series value in that point. Otherwise, LAST and VAL are both null. (The behavior for VALID and VAL is slightly different in evaluation of event conditions. The differences are explained later, in the "Condition" subsection.) A number indicates the point at the position in the window specified by the number, with the number 0 indicating the oldest point. For example, "WINDOW.0, WINDOW.1, WINDOW.2" indicates the oldest three points in WINDOW.

The payload is delivered as a pair of lists. One list contains the column names specified in the query string. The other list contains the corresponding values.

Trigger Registers

The persistent query execution system evaluates the event condition each time a point is added to any register in the set of trigger registers for the persistent query. The trigger registers syntax is as follows.

<trigger registers> = reg1 [[, reg2 [, ...]]

This is a comma-separated list of registers.

Condition

The event condition is encoded in the section of the query denoted by

<condition>

The syntax is that of a Java expression, extended to allow use of identifiers in the same REGISTER.WINDOW.NAME syntax as in the payload specification. Allowed names include statistics, VALID, and VAL. All such identifiers resolve to primitive double type values. In the expression, VALID is 1.0 if the window is valid and 0.0 otherwise. The name VAL refers to the value in the last point added to the window.

Example

For example, consider the following query string.

SELECT ABC.50.MEAN, DEF.5MIN.VAL, DEF.1MIN.ALL, DEF.50.STDEV FROM ABC, DEF WHERE ABC.50.MEAN > DEF.50.MEAN + DEF.50.STDEV

After adding the corresponding persistent query to an engine, the following occurs. After each new point is inserted into register ABC or DEF, the condition ABC.50.MEAN > DEF.50.MEAN + DEF.50.STDEV is evaluated. If the condition is true, then the payload ABC.50.MEAN, DEF.5MIN.VAL, DEF.1MIN.ALL, DEF.50.STDEV is output.

The process of converting a query string to a persistent query is called compilation. Two different compilation methods are dynamic compilation and runtime compilation. In dynamic compilation, a query parser class takes the query string as input and produces a Java program. (The query parser class translates the query into a standard form, ensures that the system can satisfy references to identifiers, and creates a fetcher class for each identifier.) Then a dynamic Java package uses the Java program to make a proxy class for the event condition evaluator. In runtime compilation, the java compiler javac and the disk are used to generate class bytes directly.

Processes

The following are step-by-step descriptions of some system processes, including details about how inter-process synchronization allows some processes to proceed concurrently. There are descriptions of processes to initialize the system, add a register, add a window, add a persistent query, and add a point.

A. Initialize the System

The system is initialized by creating a new engine instance or by loading a persisted engine instance from a file, using the following steps.

1. Create a sequencer. (The sequencer provides a unique number for each input value. The sequence numbers are used to ensure that each result is detected exactly once.)

2. Create a parser. (The parser plays a role in converting query strings to persistent queries.)

3. Set a compilation directory. (The compilation directory is used as a container for class files during runtime compilation of persistent queries.)

4. Register an event listener with the system. (As part of the event detection and payload output functions, the event listener receives notifications when the system detects true event conditions and receives output payloads.)

When de-persisting an engine from a file, initialization also contains these steps:

5. De-persist and add to the engine any registers and any windows specified by the file

6. Compile any query strings specified by the file to form persistent queries. Add the persistent queries to the engine.

After this process, the engine is ready to undertake other methods, including adding registers, windows, persistent queries, and data points. These other methods can be performed concurrently with each other after the engine is instantiated.

B. Add a Register

The method to add a register to the engine comprises the following steps:

1. Add the register to the list of engine registers.

2. Add a reference to the register to a hashtable with register names as keys. This hashtable enables fast lookup of a register by name.

3. If the register contains any windows, add each window name, in the format REGISTER.WINDOW, to an engine hashtable with window names as keys. This hashtable enables fast lookup of a window by register and window name.

C. Add a Window

The process to add a window to a register involves obtaining and releasing locks in order to ensure mutual exclusion between adding a window and parts of other processes. For each register, there are two locks, a basic lock and a booster lock. Also, each window has a lock. Obtaining a lock may involve waiting for the lock to be released by another process. Releasing a lock allows it to be obtained by another process. The process to add a window to a register comprises the following steps.

1. Obtain the basic register lock in order to ensure mutual exclusion with the process of adding a point to the register.

2. Set the number of points in the window to zero, and set window endpoints to indicate that the most recent point in the register is one position beyond the points in the window.

3. Associate the window with the register, adding the window to the list of windows to be updated for each input point received by the register.

4. Add the window name, in the format REGISTER.WINDOW, to an engine hashtable with window names as keys. This hashtable enables fast lookup of the window by register and window name.

5. Release the basic register lock.

6. To use historical data to populate the window, use the following iterative process.

6a. Obtain the register booster lock. Obtain the register basic lock.

6b. Get a reference to the next historical point to add to the window, i.e., a reference to the most recent point in the register that was added before the window and is not in the window.

6c. Obtain the window lock.

6d. Release the register basic lock. Release the register booster lock.

6e. Update the window statistics to account for the referenced point.

6f. Update the window endpoints to indicate that the referenced point is in the window.

6g. Release the window lock.

6h. Determine whether the process of populating the window is complete, as follows. For a value window, check whether the number of points in the window equals or exceeds the target. For a time window, check whether the next historical point to be added falls before the beginning of the time interval. If the process is not complete, then go to step 6a.

D. Add a Persistent Query

The engine has a query lock, which is used to ensure mutual exclusion between the process of adding a persistent query and the process of executing a persistent query. The process to add a persistent query, specified by a query string, to the engine comprises the following steps.

1. Obtain the query lock.

2. Parse the query to obtain strings corresponding to the payload specification, trigger registers, and event condition.

3. Parse the payload specification, trigger registers, and event condition strings to determine all references to registers, windows, and names. Check that the referenced entities exist. If so, then proceed. If not, then release the query lock and exit this process.

4. Compile the query string to form a persistent query, using runtime or dynamic compilation as described previously.

5. Register the persistent query with any with any trigger registers of the persistent query.

6. Release the query lock.

E. Add a Point

Each input value is accompanied by a label that indicates the time series to which the value belongs. The process by which the engine handles an input time series value comprises the following steps.

1. Use the filter to determine whether the label refers to a time series that corresponds to a register in the engine. If not then exit this process.

2. Use the sequencer to generate a unique sequence number for the value. Form a point that comprises the value and the unique sequence number.

3. Obtain the register basic lock. This ensures that no window is added to the register and that no other point is added to the register while the point is being added to the register.

4. Check whether the size of the buffer in the register is large enough to include the point in addition to any previous points in the buffer. If the buffer is not large enough, then boost the buffer size as follows.

4a. Obtain the booster lock on the register. This ensures mutual exclusion with part of the process to populate a new window with historical points.

4b. Allocate storage to increase buffer size.

4c. Copy existing points into any corresponding positions of newly allocated buffer storage.

4d. Update any window references to buffer locations.

4e. Release the register booster lock.

(A similar process to steps 4a to 4e is used to shrink the buffer size when little of the buffer is being used after data expiry or window deletion.)

5. For each window of the register, (a) obtain the window lock; (b) update the window to account for the new point; (c) determine which points (if any) have expired from the window, and update statistics to account for expiry; (d) release the window lock.

6. Discard points from the register buffer that will have no further effect on statistics or values for any window.

7. For each persistent query for which the register is a trigger, do the following. Obtain the query lock. Evaluate the event condition. If the event condition is true, then fetch and output the payload as specified by the payload specification. Then release the query lock.